TOWARDS ARCHITECTURE FOR PEDAGOGICAL AND GAME SCENARIOS ADAPTATION IN SERIOUS GAMES

Wassila Debabi[1] and Ronan Champagnat[2]
[1]*Badji Mokhtar University, Annaba, Algeria*
[2]*La Rochelle University, La Rochelle, France*

ABSTRACT

Serious games seem to be a promising alternative to traditional practices for learning. Recently, their use in computer science education and learning programming became more widespread. Actually, many students in programming courses have difficulties to master all required competencies and skills especially at introductory level and games have the potential to be an important teaching tool for their interactive, engaging and immersive activities. However, the critical point of a serious game remains the relation between the game and its pedagogical content. In fact, a serious game has to deal with two scenarios: the game scenario and the pedagogical scenario.
In this paper, we introduce a system that aims to improve algorithmic learning using serious games. We propose a method for managing both game and pedagogical scenarios along with a double adaptation: the first phase that adapts the pedagogical scenario and the second phase that adapts the game scenario. We propose a software architecture that ensures the interaction between the aforementioned two phases. The learning and the game scenarios are planned according to the student's profile, his pedagogical path, his progress in the game and also taking into account the collected traces in the observation phase.

KEYWORDS

Serious games, algorithmic, learning, adapting, motivation

1. INTRODUCTION

Data structures and algorithms are important foundation topics in computer science education. However, they are considered to be hard to teach and learn because they usually model complicated concepts, refer to abstract mathematical notions, or describe complex dynamic changes in data structures. Many students in programming courses have difficulties to master all required competencies and skill especially at introductory level. Several reasons are pointed out, and a number of researches have been carried out to recognize the characteristics of novice programmers. Mostly, the traditional learning methodologies usually based on lectures and specific programming language syntaxes, rendering learning difficult for beginners, and fail often to attract and motivate them to be implicated in programming activities (Lahtinen *et al.*, 2005; Schulte & Bennedsen, 2006). Also, programming languages typically used in programming classes are professional in nature (C, C++, C# and Java), and they have extensive and complex syntaxes, that makes learning difficult for beginners (Jenkins, 2002; Motil & Epstein, 1998). Another concern is the students' difficulties with abstract concepts: knowing how to design a solution to a problem, subdivide it into simpler sub problems, and conceive hypothetical error situations for testing and finding out mistakes (Esteves *et al.*, 2008); and difficulties in understanding even the most basic concepts (Lahtinen *et al.*, *op. cit.*; Miliszewska & Tan, 2007) such as variables, data types or memory addresses as these abstract concepts do not have direct analogies in real life (Lahtinen *et al.*, *op. cit.*; Miliszewska & Tan, *op. cit.*). Many technological solutions attempt to solve these problems by providing environments which give facility in tasks of execution and visualization mechanisms, while they failed in other aspects such as the lack of motivation and the inability to properly portray in a comprehensible way the complex computer programming concepts (Henriksen & Kölling, 2004). Recently, interest has turned to serious games as providing engaging ways of learning. Serious games technology offers tools that may have the potential to help computer-programming students become more engaged in their learning through a "learn while having fun" approach (Coelho *et al.*, 2011).

More importantly, games offer methods of learning that are highly consistent with modern theories of effective learning which recommend that learning activities should be active, situated, problem-based, interactive, and socially mediated (Boyle *et al.*, 2011).

As previously said, acquiring and developing knowledge about programming is a highly complex process, and algorithmic thinking is a term used very often as one of the most important competences that can be achieved by education in Informatics. Algorithmic thinking is somehow a pool of abilities that are connected to constructing and understanding algorithms (Futschek, 2006). Our principal attention carries on enhancing student's algorithmic thinking skill, where our goal is articulated according three aspects. The first one is to let students focusing on the resolution of the problem more than on the syntax of the programming language. The second one is to reduce their difficulties with abstract concepts: students apply human principles of thinking and acting to the computer. This includes understanding the way programs are executed, both in terms of internal variables and external files and I/O (Kaasboll, 1998). And the last one is to allow students knowing how to design a solution to a problem by dividing it into simpler sub-problems.

Another concern is the diversity of learners. This phenomenon can be met in all classrooms, where each learner is characterized by some diversity in results, abilities, interests, motivation and needs. Adaptation in serious games is an important feature and many researchers agreed that serious games and simulations have to become more challenging, unpredictable and player-centric, to be fully embraced as an effective way of knowledge transfer (Barnes *et al*, 2008; Bourse & Labat, 2012). It can also manage the players-learners' frustration while increasing their motivation.

Our goal is the contextualisation and the individualization of the gaming path, and therefore the learning path for each learner according to his/her own need. Thus, we provide the player-learner a unique game experience and effective learning experience. As a result, we are confronted to a double adaptation: the adaptation of the pedagogical scenario and the adaptation of the game scenario. To ensure this double adaptation, our game is integrated into "POLARIS" (Trillaud, 2013), a software platform that has functions to analyse users' actions and provide adaptation mechanisms. The adaptation process and the proposed architecture to ensure this feature are detailed in the following sections.

We present a brief study and discussion about existing educational games in Section 2. Section 3 introduces our proposed system. A case study is presented in Section 4. Finally, conclusions drawn from the work done so far are discussed in Section 5.

2. RELATED WORK

In order to help students and improve their learning, diverse games dedicated to learning programming were implemented. All these initiatives proposed a number of features that met the problems typically encountered in computer programming. We distinguish two categories of games, the first one aims to teach programming by asking the learner to develop his/her own game, as in **Robocode** (O'Kelly & Gibson, 2006), this game developed by IBM aspires to teach programming using Java language. The concepts taught by Robocode are the basic concepts of structured programming but also the main structures of object-oriented programming such as inheritance, polymorphism, etc. **M.U.P.P.E.T.S** (Phelps et al., 2003) "The Multi-User Programming Pedagogy for Enhancing Traditional Study" that aims to teach the basic concepts of object-oriented programming using exclusively the JAVA programming language. **EEClone** (Gestwicki & Sun, 2008), This game is an arcade-style computer game. Students analyse various design patterns within EEClone, and from this experiment, learn how to apply design patterns in their own game software. The second category includes games that ask learners to code so they can progress in the game, such as **GAME2LEARN project** (Barnes *et al., op. cit*). This project gathers two games with two distinct scenarios: "**Saving Sera**", where player must perform various tasks involving programming concepts. When the player makes a mistake, the character must fight a script bug by answering various computer science questions. The second game is "**The Catacombs**", it aims to familiarize students with activities such as variables declaration and the usage of simple as well as nested if statements and loops. Another initiative is the well-known **Hour of Code** (Code Studio, 2015) designed to demystify code and show that anybody can learn the basis using Scratch. Also **Prog & Play** (Muratet et al., 2010) a real-time strategy game, where students write programs to control units in a battlefield. They can choose the programming language amongst Ada, C, Java, OCaml, Scratch and Compalgo. **Wu's Castle** (Eagle & Barnes, 2009) is a role playing game, where students program changes in

loops and arrays in an interactive, visual way. The game provides immediate feedback and helps students visualize code execution in a safe environment. It uses the programming language C ++. **CoLoBot: Colonize with Bots** (CoLoBot, 2013) combines both a real time game of strategy and an initiation to programming. It aims to teach students Object Oriented programming style similar to C++ or Java. And **PlayLogo3D** (Paliokas *et al.*, 2011) a role playing game, especially designed for children aged 6-13 years in the early stages of programming education. PlayLogo3D aspires to introduce the very basic concepts of structured programming using LOGO programming language.

Some games listed before use macro-languages to support students' understanding the logic behind the programming elements, while others use a distinct programming language for teaching such as Pascal, JAVA, C / C ++. We believe it is a challenging situation for the learner. Dealing with the syntax of a particular programming language is an additional charge, especially for novice learners that may discourage them, leading them to withdraw the learning. Several common deficits in novices' understanding of specific programming language constructs were pointed out in many studies (Soloway & Spohrer, 1989; Jenkins, 2002) and inappropriate analogies may be drawn from natural language leading learners to serious confusion. Another concern is about actions that take place "behind the scene": the abstraction is a powerful programming concept but beginners face difficulties moving from the abstract toward the concrete (Soloway *et al. op. cit*).

Our work is based on these tools, we are inspired by the second approach but instead of asking learners to code in order to progress in the game, our approach is designed such that to bring them to take actions in the game and see the translation of their actions into algorithmic instructions, this way allows the learner to be free from the syntax complexity of programming languages, and focusing more on solving the problem. All the initiatives mentioned above propose a number of features that meet the problems typically encountered in computer programming. They introduce interesting notions such as attractive graphical interfaces, visual representation of the programming tasks, interactive and interesting scenarios that keep the learner immersed. However, these games do not take into account the learner's profile and his/her progression in the game. Adaptation in games that are either serious or not, is an important feature that allows to individualize and contextualize the gaming experience. It also allows managing the frustration of the players-learners while increasing their motivations (Hocine *et al*, 2011). Unlike video games, which are only concerned by the game aspects, adaptation in serious games must take into account the "serious" aspects related to learning goals, information or skills gaining. Therefore, adaptation must take into account not only the information collected about the player-learner during his/her interaction with the game (traces, scores achieved, progression in the game, etc.) but also the parameters related to the pedagogical goals and the goals he/she achieved by so far.

3. PROPOSED SYSTEM

The use of serious games in education is controversial. If their effects on motivation have been shown, their pedagogical potential remains questionable in the absence of a tutor's direction. (Conati *et al*, 2009) advanced as an explanation the lack of a sufficient adaptation of the program to the learner who would substitute the tutor by proposing personalized learning and activities. It should be taken into account that learners have different learning needs and styles that are difficult to define.

As a consequence, we decided to built a serious game scenario management based both on controlling the game experience and the learning progression. Our purpose is to show how to perform the interleaving decision of game and learning to progress in the game. We suggest starting from the game point of view (according to multicriteria decision algorithm as PROMETHEE II) and then to chose the best pedagogical activity (according to user profile, learning objectives and possible activities, using classification algorithm).

We propose to integrate our serious game in POLARIS platform, which ensures adaptive and dynamic execution of the learning scenario, and also the game scenario based on the user profile (who becomes a learner-player or player-learner), his/her pedagogical progression, his/her progression in the game, the score achieved in the game and the trace generated during the game session.

3.1 Constraints and Specifications

The critical point of a serious game is the relation between the game and its educational content. Several experiments have shown that serious games achieve their goals when they have a strong "game" component clearly highlighted. Thus, serious games have double script. Actually, there are two scenarios to design: a game scenario (explicit) along with another learning scenario (implicit). Our idea is that the game has n-different stages; each stage is dedicated (implicitly) to a learning goal. In each stage, the game simulates the algorithm's behavior.

Another fundamental point is how to maintain the player's motivation during the game. In fact, adaptation can significantly increase the effectiveness of the learning task by reinforcing the motivation and the learner's interest. Several approaches address this issue as (Hocine *et al*, *op.cit.*): The use of general principles and good game design practices in order to create immersion (flow); The use of appropriate and attractive human-machine interfaces in order to facilitate the player-learner's interaction and acceptance; Finally, allow a dynamic adaptation in order to individualize and contextualize the game experience for each player-learner.

In order to enhance the player's motivation and immersion, we take into account, for the development of our game, the game design specifications related in (Carron *et al.*, 2009). In addition, we focus on adaptation by proposing a system that ensures a dynamic adaptation in order to individualize and contextualize the game experience and consequently increase the satisfaction for each player-learner while improving the effectiveness of the learning.

3.2 Method

We propose in the rest of this paper a software architecture that ensures this process of adaptation and the interaction between the adaptation phases. Indeed, we proceed to a double adaptation planned in two phases: the former is "**the decision phase**" that determines the most appropriate concept to teach in the next step (adapting the pedagogical scenario). Once the concept is decided, we proceed to the second phase called "**the realization phase**" that chooses the most appropriate game mission to play (adapting the game scenario).

3.2.1 Learning Scenario

Our game is mainly dedicated to teach basic algorithmic concepts but also to improve the learner's competence with regards to algorithmic thinking and decomposition of the problem in hand. Therefore, *AlgoGame* is designed essentially for computer science students that are dealing for the first time with this area, because they have the most difficulties in understanding the theoretical concepts. Therefore, we classified the programming concepts in four categories as follows:

Fundamentals: group the very basic programming concepts such as Assignment, Instructions order, Boolean expressions, Inputs/ Outputs.

Control structures: group algorithm of Permutations, IF-THEN Statement, IF-THEN-ELSE Statement, and Loops.

Data structures: includes algorithm related to Arrays, Stacks, Queue, and Linked lists.

Further concepts: contains concepts that require the abovementioned categories to be assimilated as functions and record structures.

Some concept categories are prerequisite for learning concepts in other categories, e.g. it would be inappropriate to teach "IF-Then statement" from the Control Structures category before teaching "Boolean expressions" from the Fundamentals category. Therefore, concepts of different categories are interconnected by prerequisites. Only concepts from Fundamental category do not require any prior notion.

Our work considers the adaptation of the learning scenario (and consequently the game scenario) in an Interactive Adaptive System. We limit the interactions using contextualized blocs called situations (Pham et al. 2015). *A situation* in an interactive application is a component where actors interact using resources in a specific context to achieve the defined goals. The application execution consists in choosing, related to a given situation, the most appropriate following one (Hoang et al, 2015). In our case, a situation refers to a learning concept; there are two parameters to define: *A pre-condition*, which is a set of conditions that must be verified in order to begin a new situation (a new learning concept). *A post-condition* is the expected result after executing the situation. It represents the exit conditions from a situation.

3.2.2 The Game Scenario

AlgoGame is an exploratory game; it is designed as exploratory areas where the player must solve a mission to unlock access to new missions, which were before forbidden for access. Each mission ensures learning of one or more algorithmic concepts.

Like every game, *AlgoGame* has a set of rules that indicates how it is meant to be played by the players (the game mechanics) serving as a basis for the gameplay. The proposed core mechanics for our game are related in [4] and taking into consideration all the aforementioned constraints and specifications to meet the aimed academic goals.

3.2.3 Relation between Learning and Game Scenario

In the concern of having a coherent link between the game and its educational content, all the aforementioned learning concepts are related to one or more mission in the game. In other words, one mission in the game can be dedicated to one or more learning goals. In fact, we aim to provide an individualized gaming and learning experience to learners according to their needs. For that purpose, the learning path is planned according to **the student's profile** but also taking into account **the model of the student' scenario**, which indicates his/her progress in the game (and consequently his/her actual learning path), and the **model of learning scenarios**, which contains all the relations established between the concepts of the different categories. All these information combined to **the collected traces** in the observation phase are used in the decision phase. This phase uses a Trace-Based Subjective logic (Hoang *et al*. *op.cit*) to determine the most appropriate next concept to learn.

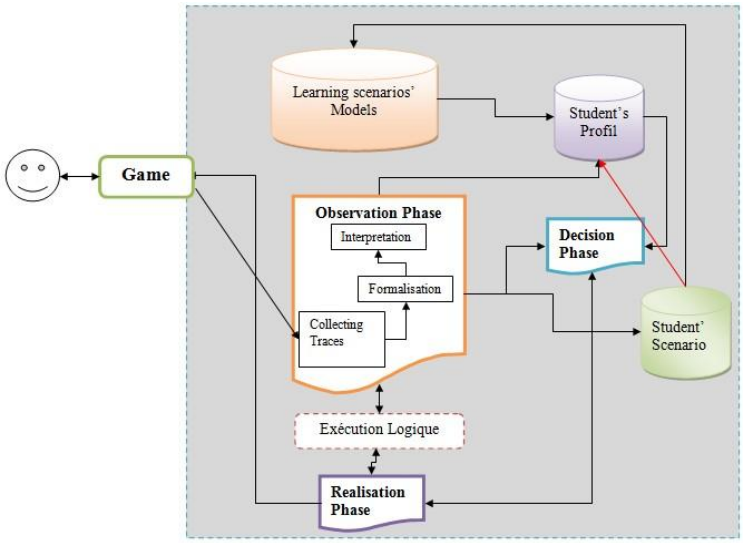

Figure 1. Software architecture

Once the learning concepts are defined, "TF-IDF" algorithm is used in the realization phase (Sawadogo, 2016) to determine the most appropriate next resource to deliver to learner and thus the next mission in the game. The figure 1 illustrates the system's architecture and the different models involved in the decision and realization phases.

4. CASE STUDY

AlgoGame aims to introduce students to algorithmic concepts, to help them get familiar with how the algorithms are structured and also enable them to understand the algorithmic thinking. The game has several levels; each one aspires to teach a programming concept from the four aforementioned categories (3.2.1), or to teach an algorithm that combines several concepts. This section will describe a case study, and we have chosen to present an illustrative example of the system functioning. This operating principle ensures an

implicit adaptation of the pedagogical scenario and also of the game scenario by providing the learner-player by the most appropriate mission to play according to his/her own needs in the current situation. We will then describe the game mission selected in our example to illustrate the programming concepts involved.

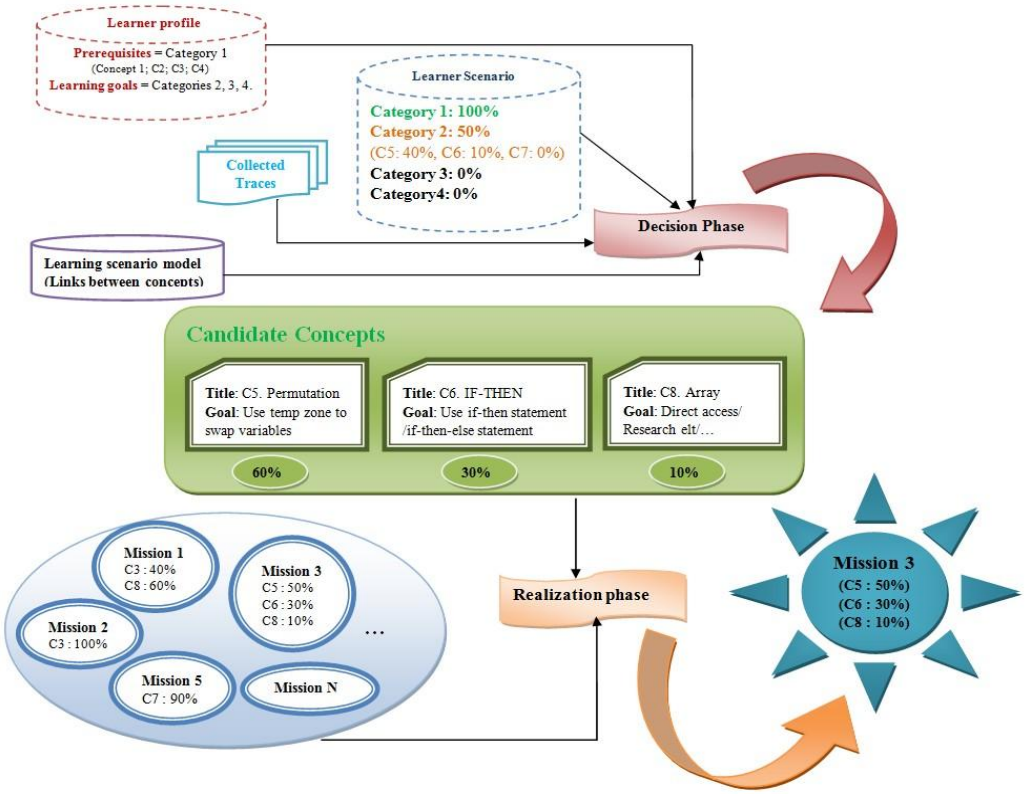

Figure 2. Case of the system's operating principles

The figure 2 represents a case of the system's operating principles. We consider a learner's profile that has the algorithmic concepts of **category1** as prerequisite and as learning goals the algorithmic concepts of the **categories 2, 3 and 4**.

Based on this learner's profile, his/her learning path (learner scenario), traces collected and the model of learning scenarios predefined in the system, the **decision phase** is launched. This phase ensures the adaptation of the pedagogical scenario. It allows deciding on the "next" most appropriate concepts to learn. We call them the "candidate concepts". Each candidate concept has a weight ranging from the most to the least relevant. In our example, the candidate concepts are: the concept **C5.Permutation** with 60%, the Concept **C6.IF-THEN statement** with 30% and the concept **C8.Array** with 10%.

Once the candidate concepts are defined, a second phase ensures the adaptation of the game scenario, it is the **realization phase**. This one is founded on the candidate concepts resulting from the decision phase as well as the game's missions predefined in the game scenario database. In our example, "Mission 3" is selected and presented to the learner as the most appropriate mission to play according to his/her evaluation in the game scenario as well as in the pedagogical scenario.

The game mission "Mission 3" introduces the selection sort algorithm, while focusing on the concept **C5.Permutation** (permutation of two variables). The concepts **C6.IF-THEN statement** and **C8.Array** are also introduced.

Gameplay: The player is placed in the Computer Science department parking; he/she has to sort, within a given time, the parked vehicles in the garages. This sorting will be done according to vehicle weight, from least heavy to heaviest. To achieve this, the learner must complete a series of specific tasks to establish the desired order before time limit; otherwise, the game ends.

The selection sort algorithm consists in sorting array elements in a given order. In order to concretize the abstract concepts related to this algorithm, we thought to materialize the array cells by garages and items by

parked vehicles within these garages. Furthermore, the vehicle weight is not communicate to the player, we suggest to indicate (graphically) only, the number of the least heavy truck, so the player has no choice other than browse all the vehicles to determine the minimum, as this goes on in the algorithm. To swap two vehicles, the player must go through an intermediate zone because the road leading to the various garages passes one vehicle at a time. This operation makes the learner get aware of the "machine" constraints, i.e. the need to go through temporary variable when permuting array's elements.

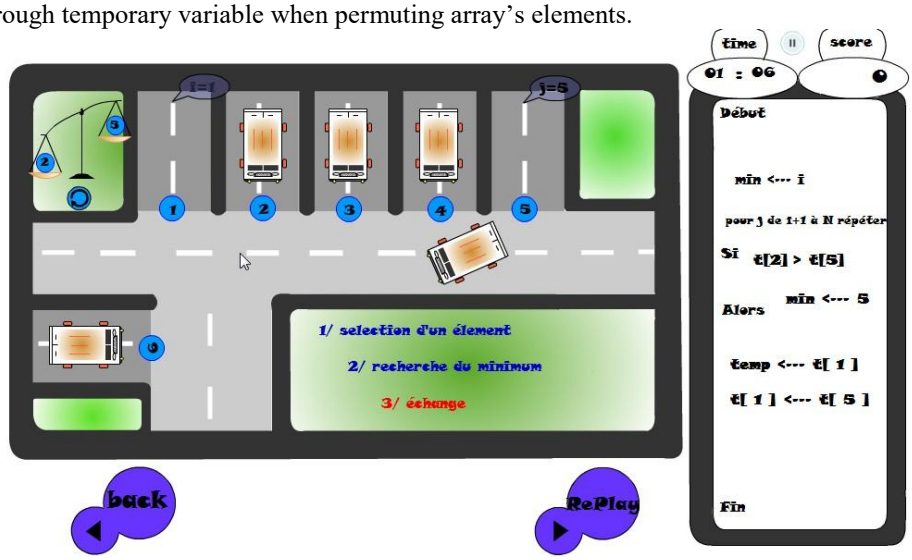

Figure 3. Mission 3 : selection sort algorithm

Every action made in the game generates an algorithmic command which appears at the right of the screen, until the whole algorithm is constructed. The game include explanatory messages that assist students in understanding the theory while playing, such as the hints with index (i) and (j). Besides, the selection sort algorithm goes by three phases: 1) Selecting an element. 2) Find the minimum or the maximum. 3) Swap the tow elements. In order to highlight these phases, the player is informed each time he/she goes through one of them.

5. CONCLUSION AND FUTURE WORKS

AlgoGame is a new solution for learning and teaching algorithmic bases that has been presented in this paper. Our principal concern was to reduce the abstraction and help students to make relation between their real world and the abstract algorithmic concepts. Also, bring them to realize that a problem can be divided into simpler sub problems by highlighting the algorithm's steps when the player goes through them. The game discharge student from syntax and debug errors, thus the algorithms are not constructed by typing text or arranging icons but by taking actions in the game.

Another important aspect is the diversity of learners, the adaptation of the learning scenario (and consequently the game scenario) according to learners needs was a main concern for us, thus we propose to integrate our game in the adaptive platform POLARIS. The proposed architecture uses different modules to ensure the adaptation of both pedagogical and game scenarios according to students needs. The particularity of serious games is the double point of view "*game*" and "*serious*". We have highlighted the need to have an adaptation for each one: adapting the progress of the game and adapting the pedagogical sequence. To do this we assume that it is better to decouple adaptation in two stages. Firstly, the adaptation of the pedagogical scenario and then the adaptation of the play activity, we then showed how to make the interaction in the piloting of each scenario.

So far, we chose to verify the usability and effectiveness of the game apart. We have analyzed the impact of introduction of gaming in the programming activity. The first evaluation results are encouraging and motivating. In a second step, we aim to integrate the game into POLARIS platform. This will involve another controlled experiment to study whether the use of this system has any impact of the student programming experience, and if so, whether these effects are positives or negatives.

REFERENCES

Barnes, T. et al. (2008). Game2Learn: improving the motivation of CS1 students. In *Proceedings of the 3rd international conference on Game development in computer science education* (pp. 1-5).

Bourse, Y., Labat, J.M. *Adaptation sur la progression de l'apprentissage dans les jeux sérieux*. Dans IREC Conference 2012.

Boyle, E. et al. 2011. The role of psychology in understanding the impact of computer games. *Entertainment Computing*, *2*(2), 69-74.

Carron, T. et al. 2009. How to bring immersion into Learning Games?. In *Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference*, pp. 358-360.

Coelho, A. et al. 2011. Serious game for introductory programming. In *International Conference on Serious Games Development and Applications* (pp. 61-71). Springer Berlin Heidelberg.

Code Studio: https://studio.code.org/. Accessed: 28 August 2015.

CoLoBoT 2013: http ://www.ceebot.com/colobot/index-e.php. Accessed 18 September 2013.

Conati C., Manske M., 2009. Evaluating Adaptive Feedback in an Educational Computer Game, Proceedings of the 9th International Conference on Intelligent Virtual Agents, 2009, Amsterdam, The Netherlands.

Eagle, M., Barnes, T. 2009. Experimental evaluation of an educational game for improved learning in introductory computing. *ACM SIGCSE Bulletin* 41(1), 321-325.

Esteves, M, et al. 2008. Contextualization of programming learning: A virtual environment study. In *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual* (pp. F2A-17). IEEE.

Futschek, G. 2006. Algorithmic thinking: the key for understanding computer science. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 159-168). Springer Berlin Heidelberg.

Gestwicki, P., Sun, F.S. 2008. Teaching Design Patterns through Computer Game Development. Journal on Educational Resources in Computing (JERIC), vol. 8(1), March 2008.

Henriksen, P., & Kölling, M. 2004. Greenfoot: combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pp 73-82.

Hoang, N.H. et al. 2015. Application of Trace-Based Subjective Logic to User Preferences Modeling. 20th International Conference on Logic Programming, Artificial Intelligence and Reasoning, EPiC Series, vol. 123, pp. 80 – 90.

Hocine, N. et al. 2011 *Techniques d'adaptation dans les jeux ludiques et sérieux.* Revue d'Intelligence Artificielle, Lavoisier (Hermes Science Publications), 2011, 25 (2), pp.253-280.

Jenkins, T. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Vol. 4, No. 2002, pp. 53-58.

Kaasbøll, J. J. 1998. *Exploring didactic models for programming*. In NIK 98– Norwegian Computer Science Conference. p.p. 195-203.

Lahtinen, E. et al, 2005. A study of the difficulties of novice programmers. In *ACM Sigcse Bulletin*, Vol. 37, No. 3, pp. 14-18.

Miliszewska, I., & Tan, G. 2007. Befriending computer programming: A proposed approach to teaching introductory programming. *Informing Science: International Journal of an Emerging Transdiscipline*, *4*(1), 277-289.

Motil, J., & Epstein, D. 1998. JJ: a language designed for beginners (less is more). *Available at http://www.publicstaticvoidmain.com*.

Muratet, M. et al. 2010. Experimental feedback on Prog&Play, a serious game for programming practice, *Eurographics, pp.1-8.*

O'Kelly, J. & Gibson, J. P. 2006, RoboCode & problem-based learning: a non prescriptive approach to teaching programming, *in: Proceedings of Conference on Innovation and Technology in Computer Science Education. 217–221.*

Paliokas, I. et al. 2011, *PlayLOGO 3D: A 3D interactive video game for early programming education, in:* Proceedings of 3rd International Conference on Games and Virtual Worlds for Serious Applications. 24-31.

Pham, P.T. et al. 2015. A Situation–Based Multi–Agent Architecture for Handling Misunderstandings in Interactions. International Journal of Applied Mathematics and Computer Science, De Gruyter, 2015, 25 (3), pp.439-454.

Phelps, A.M. et al. (2003), MUPPETS: multi-user programming pedagogy for enhancing traditional study, *in: Proceedings of 4th conference on Information technology education. 100-105.*

Schulte, C. & Bennedsen, J. (2006), *what do teachers teach in introductory programming? In*: Canterbury, UK, 2ed International Workshop on Computing Education Research. Pp 17–28.

Soloway, E. & Spohrer, J. 1989. *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale, New Jersey. p.497.

Sawadogo, D. 2016. Thesis : *Architectures logicielles et mécanismes pour la gestion adaptative et consolidée de ressources numériques dans une application interactive scénarisée.*

Trillaud, F. 2013. Thesis: *Architecture for interactions control and interactive and adaptive multiusers applications driving: application to e-learning*. Rochelle University. https://tel.archives-ouvertes.fr/tel-01140071